OFFICIAL

PATENT

MNFRAME.041A

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	:	Liu, et al.	Group Art Unit 2785
Appl. No.	:	08/942,384	
Filed	:	October 1, 1997	
For	:	SYSTEM FOR AUTOMATICALLY REPORTING A SYSTEM FAILURE IN A SERVER	
Examiner	:	L. Hua	

DECLARATION UNDER 37 C.F.R. § 1.131

1. This declaration is to establish the status of the invention in the above-captioned U.S. patent application in the United States on May 30, 1996, which is the effective date of U.S. Patent No. 5,815,652, entitled "COMPUTER MANAGEMENT SYSTEM", to Ote, et al. which was cited by the Examiner against the above-captioned application.

2. We are the named joint inventors of the described subject matter and all claims in the above-referenced application.

3. We have read the Office Action mailed October 27, 1999 (Paper No. 11) regarding the patent application.

4. We developed our invention as described and claimed in the subject application in this country, and acted with due diligence to reduce the invention to practice from at least May 30, 1996, as evidenced by the following events:

a. By at least May 30, 1996, we had conceived of a system for reporting a failure condition in a server system, comprising: a controller which monitors the server system for system failures and generates an event signal and failure information if a system failure is detected, a system interface, coupled to the controller, which receives the event signal, a central processing unit, coupled to the system interface, wherein, upon receiving the event signal, the

system interface reports an occurrence of an event to the central processing unit, and a system log which receives failure information communicated from the system interface and stores said failure information.

b. Furthermore, by at least May 30, 1996 we had conceived of a failure reporting system for a server system, comprising a controller which monitors the server system for system failures and generates an event signal and failure information if a system failure is detected, a system recorder, coupled to the controller, which receives the failure information and assigns a date and time to the failure information, a system log which stores the failure information, a system interface, coupled to the controller, which receives and stores the event signal and reports an occurrence of an event to a central processing unit, coupled to the system interface, wherein the central processing unit executes a software program which allows a system operator to access the system log to read failure information stored therein, a remote interface, coupled to the controller, which receives the event signal and reports the occurrence of an event to a computer external to the server system, and a switch, coupled to the remote interface, which switches connectivity to the remote interface between a first computer and a second computer, wherein the first computer is a local computer, coupled to the switch via a local communications line, and the second computer is a remote computer, coupled to the switch via a modem connection.

c. Our conception and subsequent inventive activity leading to an actual and constructive reduction to practice of an embodiment of the present invention is evidenced by the following:

d. By at least May 30, 1996, we had conceived of using a network of microcontrollers as the monitoring and control hardware of the subject invention. A document, entitled "Raptor Wire Service Architecture, Version 1.2" ("Wire Architecture"), was written at least as early as March 19, 1996, as evidenced by the document date. A copy of Wire Architecture is attached as **Exhibit A**. Wire Architecture describes a controller which monitors the server system for system failures, and generates an event signal and failure information if a system failure is detected (CPU A and/or CPU B controller shown on figure on page 1), a system interface, coupled to the controller, which receives the event signal (system interface controller shown on figure on page 1), a central processing unit, coupled to the system interface, wherein, upon receiving the event signal, the system interface reports an occurrence of an event to the

central processing unit (system interface controller connected to central processing unit via ISA bus; "it is assumed that the interface is reliable between the system processors and the interface processor.", p. 13; "If any data temperature exceeds limit and WS_SYS_OVERTMP is clear, set WS_SYS_OVERTEMP, send a TEMPERATURE event to the system and remote interfaces, log WS_SYS_TEMP_SHUT", p. 33), and a system log which receives failure information communicated from the system interface and stores said failure information (NVRAM shown in Figure 1 and connected to system recorder).

Wire Architecture also describes in addition to the features described above, a controller which monitors the server system for system failures and generates an event signal and failure information if a system failure is detected, a remote interface which receives the event signal and reports the occurrence of an event to a computer external to the server system (remote interface controller shown on Figure 1).

e. We had conceived of a control diagnostic and monitor subsystem for a server system. A document, entitled "Raptor System: A Bird's Eye View, Version 0.99", was written at least as early as November 2, 1995, as evidenced by the document date. A copy of the cover page, and pages 8 and 9 of document is attached as **Exhibit B**. The Control Diagnostic and Monitor subsystem (CDM) supervises or monitors various system attributes, such as environmental conditions. In one embodiment, the CDM creates a request message which identifies one or more environmental conditions of the computerized environment, sends the request message from a requestor to a microcontroller network which manages the environmental conditions, obtains the status of the identified conditions, creates a response message reporting the status, and sends the response message from the microcontroller network to the requestor.

Further, Exhibit B, page 8, describes a system to monitor and manage specific functions of the first computer through a Control Diagnostic and Monitor (CDM) subsystem implemented by distributed CDM microprocessors connected to an I^2C serial (CDM) bus. The CDM can supervise and manage selected environmental conditions externally from a remote second computer via the CDM bus and communication lines. Examples of monitored and managed environmental conditions of a computer are fan speed, the temperatures of the motherboard, and of the backplane. Thus, Exhibit A illustrates providing a microcontroller network and executing

Appl. No. : 08/942,384
Filed : October 1, 1997

commands on at least one microcontroller which manages and diagnoses system functions (Claim 1).

5. I, Karl S. Johnson, am listed as an inventor on provisional Patent Application Nos. 60/046,397, 60/047,016, 60/046,416, each filed May 13, 1997, which each are priority applications for the subject application.

6. We are the listed inventors on the subject regular patent applications filed on October 1, 1997.

7. All acts leading to the reduction of practice were performed in the United States.

8. This declaration is submitted prior to a final rejection.

Penalty of Perjury Statement

We declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful, false statements may jeopardize the validity of the application or any patent resulting therefrom.

Dated: MARCH 7, 2000

By: Karl S. Johnson
Karl S. Johnson

Dated: _____

By: _____
Ji-Whan Liu

Dated: _____

By: _____
Ken Nguyen

S:\DOCS\EMN\EMN-4426.DOC
012500

Appl. No. : 08/942,384
Filed : October 1, 1997

commands on at least one microcontroller which manages and diagnoses system functions (Claim 1).

5. I, Karl S. Johnson, am listed as an inventor on provisional Patent Application Nos. 60/046,397, 60/047,016, 60/046,416, each filed May 13, 1997, which each are priority applications for the subject application.

6. We are the listed inventors on the subject regular patent applications filed on October 1, 1997.

7. All acts leading to the reduction of practice were performed in the United States.

8. This declaration is submitted prior to a final rejection.

Penalty of Perjury Statement

We declare that all statements made herein of our own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful, false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful, false statements may jeopardize the validity of the application or any patent resulting therefrom.

Dated:_____ By:_____
Karl S. Johnson

Dated: Mar. 08, 2000 By: Ji-Whan Liu
Ji-Whan Liu

Dated:_____ By:_____
Ken Nguyen

S:\DOCS\EMN\EMN-4426.DOC
012500

EXHIBIT A

Raptor Wire Service Architecture

Version 1.2

3/19/96

Prepared for
NetFrame Raptor Implementation Group

by
Karl Johnson (KJ)

Table of Contents

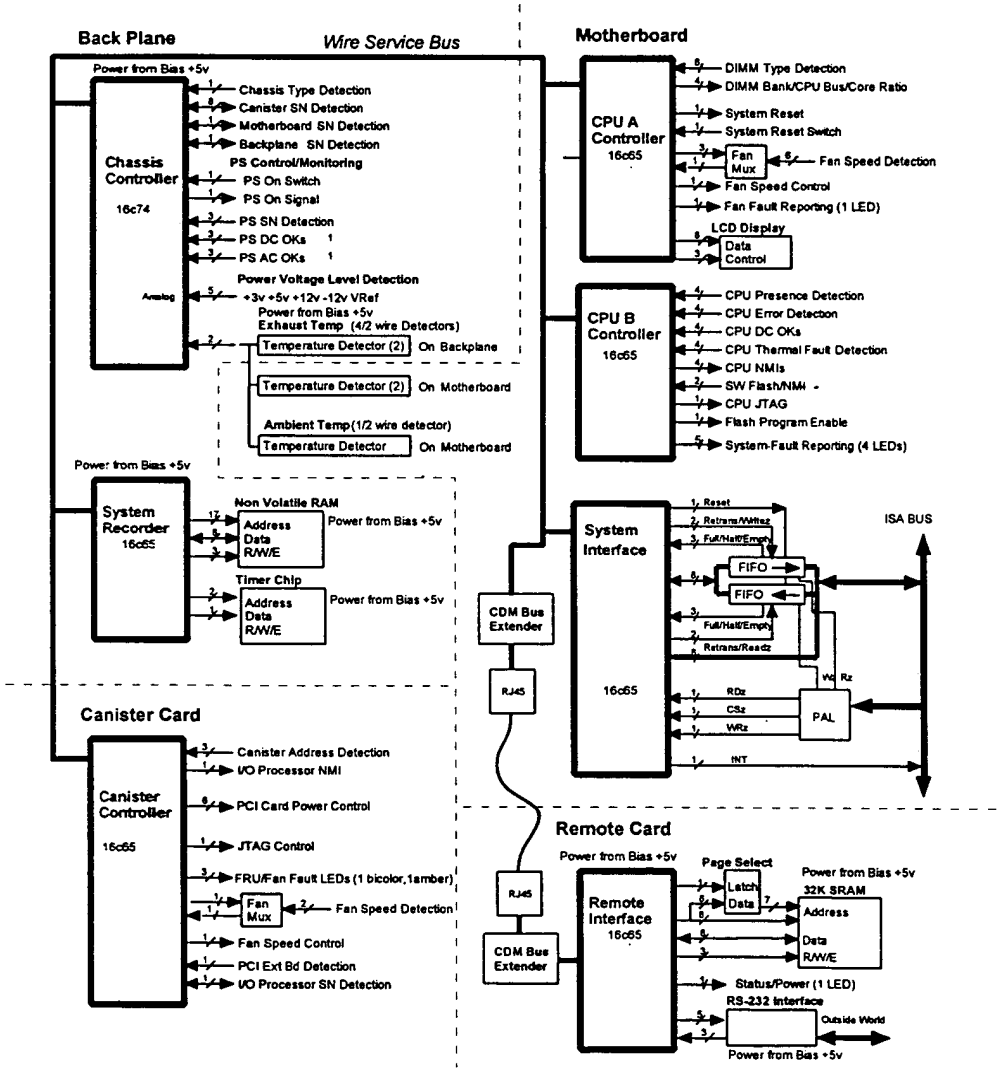
Introduction

"Wire Service" is the code name for the Raptor project system control, diagnostic and maintenance bus (formerly known as the CDM bus). Raptor is a completely "fly by wire" system - no switch, indicator or other control is directly connected to the function it monitors or controls, instead all the control and monitoring connections are made by the network of processors that comprise the "Wire Service" for the system. The processors are Microchip PIC processors and the network is a 400 kbps I^2C serial bus. A limited understanding of I^2C protocol is a prerequisite for understanding Wire Service protocols (See "The I^2C-bus and how to use it" - Philips Semiconductor, Jan 1992). Control on this bus is distributed, each processor can be either a master or a slave and can control resources on itself or any other processor on the bus.

Introduction

"Wire Service" is the code name for the Raptor project system control, diagnostic and maintenance bus (formerly known as the CDM bus). Raptor is a completely "fly by wire" system - no switch, indicator or other control is directly connected to the function it monitors or controls, instead all the control and monitoring connections are made by the network of processors that comprise the "Wire Service" for the system. The processors are Microchip PIC processors and the network is a 400 kbps I^2C serial bus. A limited understanding of I^2C protocol is a prerequisite for understanding Wire Service protocols (See "The I^2C-bus and how to use it" - Philips Semiconductor, Jan 1992). Control on this bus is distributed, each processor can be either a master or a slave and can control resources on itself or any other processor on the bus.

Wire Service Hardware Block Diagram



Logical Model

All of the command, diagnostic, monitoring and history functions are accessed using a global network memory model. That is, any function may be queried simply by generating a network "read" request targeted at the function's known global network address. In the same fashion, a function may be exercised simply by "writing" to its global network address. Any Wire Service processor may initiate read/write activity by sending a message on the I^2C bus to the processor responsible for the function (which can be determined from the known global address of the function). The network memory model includes typing information as part of the memory addressing information. It implements separate address spaces for each data type. This allows for compact internal storage and creates unique, more complex, data types which better suited to specific functions.

Message Protocol

Using a network global memory model places relatively modest requirements for the I^2C message protocol.

- All messages conform to the I^2C message format including addressing and read/write indication.
- All I^2C messages use 7 bit addressing and the "General Call" address is not used.
- Any processor can originate (be a Master) or respond (be a Slave)
- All message transactions consist of I^2C "Combined format" messages. This is made up of two back to back I^2C simple messages with a repeated START condition between (which does not allow for re-arbitrating the bus). The first message is always a Write (Master to Slave) and the second message is a Read (Slave to Master).
- Only two types of transactions are used: Memory-Read and Memory-Write.
- Sub-Addressing formats vary depending on data type being used.

Generic Wire Service I²C Message Format

The generic Wire Service message format is designed for easy encode/decode and reliability. It is not necessarily always the most compact representation possible

Master Asserts START

Offset	MSB	LSB	
Byte 0	Slave Address	0	0 means I²C write to slave
Byte 1	Data Type	R/W	R/W = 0 Mem Write / 1 Mem Read
Byte 2	Sub-Address		Least Significant Byte of Address
Byte 3	Sub-Address (Continued)		Most Significant Byte of Address
Byte 4	Length of Data (L)		Write length or read max (Note 1)
Byte 5	Data Byte 1		Present only for memory write
:	:		:
Byte N	Data Byte L		
Byte N+1	Check Byte		For data integrity (Note 2)

Master repeats START

Byte 0	Slave Address (same)	1	1 means I²C read from slave
Byte 1	Length of Data (L)		Length of data (if any)
Byte 2	Data Byte 1		
:	:		
Byte N	Data Byte L		
Byte N+1	Status		Status 0=Success otherwise Failure
Byte N+2	Check Byte		For data integrity (Note 2)

Note 1: On a memory write, this is the length in bytes of the data immediately following (not including status and check byte). On a memory read, this specifies the maximum number of bytes of data that can be returned (not including status and check byte).

Note 2: The check byte is only appended on actual I²C messages on the Wire Service Bus. It

guarantees integrity of a message on the wire. It is not required when sending messages through the system bus interface because it is automatically appended or checked by the interface.

Data Type Descriptions

Each data type description includes a rational for its existence and an example simple protocol message. The check byte is not shown in the following message descriptions for clarity. Only the message description common to all views of Wire Service is shown, although the check byte will be appended and checked by all messages as they are physically sent on the Wire Service Bus.

Bit Type

The bit data type is to be used for simple logic valued items (TRUE/FALSE, ON/OFF, etc.)

Read Bit Message
Request

Slave Address	Type/RW	Bit Addr LSB	Bit Addr MSB	Request 1

Response

Slave Address	Length 1	Bit Value (0/1)	Status 0/Success

Write Bit Message
Request

Slave Address	Type/RW	Bit Addr LSB	Bit Addr MSB	Length 1	Bit Value (0/1)

Response

Slave Address	Length 0	Status 0/Success

Byte Type

The byte data type is to be used for single byte valued items (0-255)

Read Byte Message
Request

Slave Address	Type/RW	Byte Addr LSB	Byte Addr MSB	Request 1

Response

Slave Address	Length 1	Byte Value	Status 0/Success

Write Byte Message
Request

Slave Address	Type/RW	Byte Addr LSB	Byte Addr MSB	Length 1	Byte Value (0-255)

Response

Slave Address	Length 0	Status 0/Success

String Type

The String type is designed to handle data that is best organized as variable length strings of 0 to 255 bytes. Internal allocation of string storage for each string may be less than 255 bytes, and writing a string longer than available storage will return an error.

Read String Message
Request

Slave Address	Type/RW	String Addr LSB	String Addr MSB	Request 0-255

Response

Slave Address	Length N	String Data 1	. . .	String Data N	Status 0/Success

Write String Message
Request

Slave Address	Type/RW	String Addr LSB	String Addr MSB	Length N	String Data 1	. . .	String Data N

Response

Slave Address	Length 0	Status 0/Success

Lock Byte Type

The lock byte data type is to be used for single bytes of data used to control synchronization as it performs a test and set operation on the byte. The byte is unconditionally set to zero and the original value of the byte is returned. This data type shares the same address space with the Byte Array data type.

Lock Byte Message (Read Type)
Request

Slave Address	Type/RW	Byte Addr LSB	Byte Addr MSB	Request 1

Response

Slave Address	Length 1	Original Byte Value	Status 0/Success

Byte Array Type

The byte array data type is to be used for general storage of data that is unanticipated in this architecture. This storage is only implemented in the Wire Service processor with NVRAM. External code will be responsible for managing allocation/deallocation and data directory information. Recommend that byte at address 0 be the lock byte for modification of the data directory.

Read Byte Array Message
Request

Slave Address	Type/RW	Start Addr LSB	Start Addr MSB	Request 1-255

Response

Slave Address	Length N	Array Data 1	. . .	Array Data N	Status 0/Success

Write Byte Array Message
Request

Slave Address	Type/RW	Start Addr LSB	Start Addr MSB	Length N	Array Data 1	. . .	Array Data N

Response

Slave	Length	Status

Address	0	0/Success

Log Type

The Log data type is to be used for logging byte strings in circular log buffer. It is used to record system events in the NVRAM system log.

Read Log Message
Request

Slave Address	Type/RW	Log Addr LSB	Log Addr MSB	Request 1-255

Response

Slave Address	Length N+6	Log Time LSB	Log Time	Log Time	Log Time MSB	Log Addr LSB	Log Addr MSB
Log Data Byte 1	. . .	Log Data Byte N	Status 0/Success				

Write Log Message
Request

Type/RW	N/A	N/A	Length N	Log Data Byte 1	. . .	Log Data Byte N

Response

Slave Address	Length 0	Status 0/Success

The addressing of log entries has some special characteristics:

- Reading address 65565 (0xffff) is special - It represents the address of the latest entry in the log.
- Reading address 65564 (0xfffe) is also special - It represents the address of the earliest available entry.
- The address of real log entries wraps at 65519 (0xffef). The next sequential entry after 65519 is 0.
- The address of is ignored on write and the next available entry is written.
- To read the entire log in forward time order, read entry at address 65564. This returns the first log entry along with its actual log address. Increment that address by one and read that entry. Repeat the last step until status indicates failure.
- To read the entire log in reverse time order, read entry at address 65565. This returns the last log entry along with its actual log address. Decrement that address by one and read that entry. Repeat the last step until status indicates failure.
- To keep a complete external copy of the log, first read the entire log in forward time order and

remember the last valid entry. Then periodically read forward from the remembered last valid entry to the end and add that to the external copy.

There are also conventions for the Log Data portion of the message or response. These are conventions and are not enforced by the Wire Service logging function. The conventions are as follows:

Log Data Byte 1: Severity Level Byte
- 0x00 - Unknown
- 0x10 - Informational
- 0x20 - Warning
- 0x30 - Error
- 0x40 - Severe/Fatal Error

Log Data Byte 2: Source/Encoding Byte - which entity logged the entry in the 4 high bits
- 0x00 - Wire Service Internal
- 0x10 - Onboard Diagnostics
- 0x20 - External Diagnostics
- 0x30 - BIOS
- 0x40 - Time Synchronizer
- 0x50 - Windows
- 0x60 - Windows/NT
- 0x70 - NetWare
- 0x80 - OS/2
- 0x90 - UNIX
- 0xA0 - VAX/VMS

- which type of encoding of the message is used in the 4 low bits of the byte.
- 0x00 - Binary
- 0x01 - ASCII
- 0x02 - Unicode

Thus an external diagnostics ASCII error message would have a severity and source/encoding byte values of 0x30 and 0x21.

For binary encoded messages, additional conventions apply:

Log Data Byte 3 and 4 are used as the LSB and MSB of a 16 bit Message Identifier. Each source group (OS's, BIOS, etc..) that wishes to log binary messages must maintain a file in a common format containing the definition of ALL possible binary messages from their source. This file contains the Identifier value and formatting string and any descriptive comments for each message. Identmust be maintained by each possible message source group and supplied as a file to any requestor. (Obsolete Message Identifiers MAY NOT be reused for old version compatability reasons) Log Data Bytes 5 and beyond are message arguments in format list order.

An example binary message definition file entry for the BIOS might be:
00045 "DIMM configuration bad - row %db - Types %3.3db %3.3db %3.3db %3.3db"

Note: Most "C" printf formatting items will work except that the final letter of a numeric formatting sequence must be b (byte) s (short) or l (long) to how many bytes of the log message data it takes. Strings take up to the first null (zero terminated).

Event Type

The event data type is to be used for alerting external interfaces of events in the Wire Service network. Event memory is organized as a bit vector. Each bit in the vector represents a particular type of event and there are a minimum of 32 bits in the vector.. Writing an event sets the bit representing the event in the bit vector. Whenever any bit in the event bit vector is non-zero, the interface indicates that events are present..
Reading the event type returns the complete event bit vector as a byte string of four (4) bytes with bit 0 being the low order bit of the first byte of the byte string. Once a read events completes successfully, the bit vector is cleared of event indications automatically.

Read Events Message
Request

Slave Address	Type/RW	N/A	N/A	Request 1-255

Response

Slave Address	Length N(4)	Events 1-8	. . .	Events 25-32	Status 0/Success

Write Event Message
Request

Slave Address	Type/RW	N/A	N/A	Length 1	Event ID (1-32)

Response

Slave Address	Length 0	Status 0/Success

Possible Event Types:
CPU Status Change

Power Status Change
Canister Status Change
Fan Status Change
Communications Queue Event

Screen Type

The screen data type is to be used for communication of character mode screen information from the system BIOS to remote management interface.

Read Screen Message
Request

Slave Address	Type/RW	S Addr LSB	S Addr MSB	Request 1-255

Response

Slave Address	Length N	Screen Data 1	. . .	Screen Data N	Status 0/Success

Write Screen Message
Request

Slave Address	Type/RW	S Addr LSB	S Addr MSB	Length N	Screen Data 1	. . .	Screen Data N

Response

Slave Address	Length 0	Status 0/Success

The screen address space consists of an image of character video memory for a 80x50 screen. Each character cell has both a character byte and attribute byte for a total of 8000 bytes of screen memory. Additionally memory is implemented up to address 8191. Use of the bytes above 8000 are defined by the BIOS and the remote managment software. Addresses 8001 and 8002 are suggested as the cursor address register.

Queue Type

The Queue type is designed to move data between the system and a remote management software. Queues are written/read in FIFO order and no access is available to any elements except to read the first and write the last. Queue elements are variable length up to 255 bytes. If there is no room to add an element to a queue, the entry is not added and failure status is returned. If there is no queue element available to read, failure status is returned. Interpretation of data in the queue elements is left to external software.

Note: The Queue ID in the normal address field refers to which Queue of several possible queues and NOT the queue element address.

Read Queue Message
Request

Slave Address	Type/RW	Queue ID LSB	Queue ID MSB	Request 0-255

Response

Slave Address	Length N	Queue Data 1	. . .	Queue Data N	Status 0/Success

Write Queue Message
Request

Type/RW	N/A	N/A	Length N	Queue Data 0	. . .	Queue Data N

Response

Slave Address	Length 0	Status 0/Success

System Bus Interface

Introduction

The System Bus Interface to Wire Service is the interface used by any of the system processors to read or write Wire Service memory by sending Wire Service messages and receive responses. The Wire Service messages and responses are exactly as described in the previous section (Note: The system processor is not required to compute and add or process check bytes on the message as the interface processor does that function). It is assumed that interface is reliable between the system processors and the interface processor..

The System Bus interface to the Wire Service I^2C bus is implemented by a dedicated Wire Service System Interface Processor (WSSIP)and 2 message FIFOs, one for message data written by the system processors to the System Bus Interface (requests) and one for message data returned by the system bus interface (responses). The system bus interface appears in system I/O space as 2 registers, each 8 bits wide. The lower register is the message data register (MDR) and the upper register is the command and status register (CSR).

The MDR, when written from the system bus, loads a byte into the request FIFO and when read from the system bus presents a byte unloaded from the response FIFO if any. The FIFOs may be most quickly loaded and unloaded via REP OUT or REP IN instructions executed on the system processor(s). The protocol is designed to load and unload these FIFOs with a minimum of interpretation of the message data (i.e. all message fragments have length fields in the same position and same interpretation). These FIFOs serve two functions: 1) They match speeds between the very fast System Bus and the slower WSSIP and I^2C bus and 2) They temporarily serve as interim memory for the messages relieving the Wire Service processor of that requirement.

The CSR is implemented with a direct interface to the WSSIP. Writing to this register from the system bus interrupts the WSSIP and may cause it to take actions depending on the value written. One action could be to change the value returned if there is a read from the system bus to this register. Note that there is no hardwired relation between values written to and read from this register and also that there is a processing interval between writing from the system bus to this register and the WSSIP reading the register value written. This could lead to data loss should the system bus write a second time to the CSR before the WSSIP read the first item. To avoid this problem, all writes to the CSR eventually result in some change to the value read from the CSR. Thus after writing the CSR, you should wait to see the change in the CSR indicating it has processed the data written to it.

Interface Operation

Operation of the system bus interface is controlled through the CSR.

CSR Write Format

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Int Req	Command Value						

Field Descriptions
Command Value - The value corresponding to the command to execute.
Int Req - Commands are modified by this bit to request an interrupt upon completion.

CSR Read Format

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Int Pend	Int Ena	Events	Done		Allocated To		

Field Descriptions
Allocated - The interface is currently in use. Only Allocate commands will be accepted until set.
Done - A command has completed (except for Allocate, Deallocate and Clear Done).
Events - The Event Queue on the WSSIP is not empty if this bit is set.
Int Ena - Interrupt on Events bit going from 0 to 1 is enabled.
Int Pending - An interrupt source is active.

CSR commands that are currently defined and their actions:

Commands	Value	Actions
Allocate	1-7 0x01-07	Clears both FIFOs of any stale data, clears Done and then sets Allocated To Identifer field to lower 3 bits of allocate command value. Only executed if Allocated To field is zero. Generally, first command of sequence.
Deallocate	16 0x10	Clears Done and Allocated. Generally the last command of transaction sequence.
Enable Ints	17 0x11	Enables interrupts on Events bit being set. Sets Done. May cause interrupt if Events is already set.
Disable Ints	18 0x12	Disables interrupts on Events bit being set. Sets Done.
Message	19 0x13	Request in request FIFO is sent on the I^2C bus and the response is received into response FIFO. Sets Done. I^2C bus errors create unique error response.
Clear Done	20	Clears the Done bit.

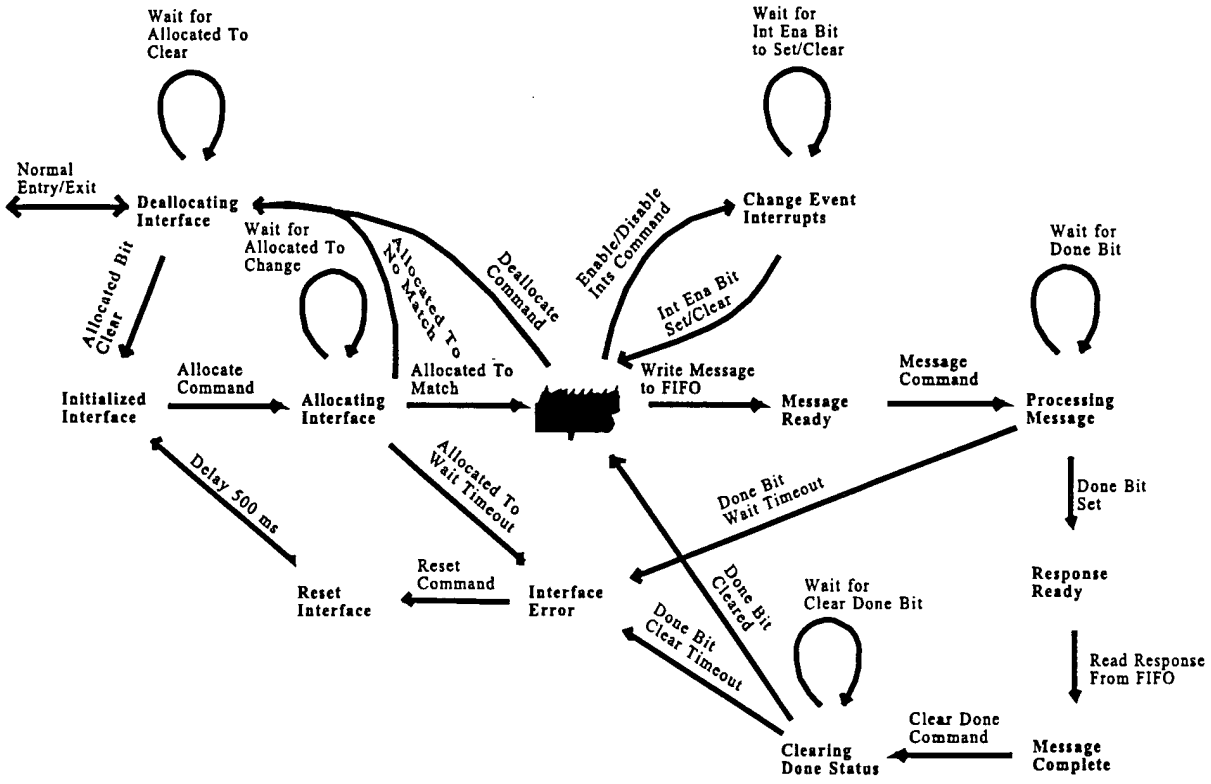
	0x14	
Reset	165 0xa5	Unconditionally clears all bits in the Read CSR except "Events". Aborts any currently in progress message operation and clears any interrupt.

Note: The purpose of the Allocate command and Allocated To field in the CSR is to provide for some degree of coordination in the case where multiple threads (or processors) are each trying to use the interface. The proper use of the Allocate command is to assign a different allocation identifer (presented in the lower three (3) bits of the Allocate command) to each possible conflicting allocation thread of control. Allocate commands will only succeed if the Allocated To field of the CSR is zero. If more than one thread issues an Allocate command at near the same time, only one will be recognized and the low order three bits of that command will be placed in the Allocated To field indicating the which thread has allocated the interface sucessfully. The losing thread realizes that the Allocated To field is not zero and not its ID, so it must give-up and try to allocate the interface later.

The following is a state diagram that details the expected use of the Wire Service system interface.

Not every possible error condition and possible state transition is anticipated by the above diagram, but it does indicate the intended operational sequence to be used with the interface in non-interrupt driven mode. To extend the diagram to interrupt mode, simply add the interrupt request bit to each command desired and instead of waiting for the appropriate bit, exit and wait for the interrupt. In most cases, the only command in which it makes sense to request an interrupt is the Message command. All other commands either complete very quickly or ignore the interrupt bit (Reset Command).

Note: The Reset Command is a problem determining if it is complete as it sets no bit and may take some time to work because of existing activity, so it seems the only way to do it now it to set it and wait.



Wire Service Network Physical Connections

The following table describe all of the physical signal connections to all of the Wire Service processors. The names for the connections will be related to network accessible memory data in the section which follows called "Wire Service Network Memory Map".

Note: All signal types and definitions are from the viewpoint of the individual Wire Service PIC processor
(e.g. Input means input to PIC processor)

Wire Service System Bus Interface (System Type ID: S0) Processor ID 10

Pin	Type	Name	Function	Notes
RA0	I	S0_FIFO_IEFZ	In FIFO (ISA Writes) Empty Flag (Active Low)	Status Flags for both ISA bus FIFO's. Need to monitor for incoming message overruns. Also must assure that output FIFO is empty before loading output message.
RA1	I	S0_FIFO_IHFZ	In FIFO (ISA Writes) Half-full Flag (Active Low)	
RA2	I	S0_FIFO_IFFZ	In FIFO (ISA Writes) Full Flag (Active Low)	
RA3	I	S0_FIFO_OEFZ	Out FIFO (ISA Reads) Empty Flag (Active Low)	
RA4	I	S0_FIFO_OHFZ	Out FIFO (ISA Reads) Half-full Flag (Active Low)	
RA5	I	S0_FIFO_OFFZ	Out FIFO (ISA Reads) Full Flag (Active Low)	
RB0	I/O	S0_FIFO_D0	ISA FIFOs Data bus Bit 0	This is an 8 bit bi-directional port for the ISA FIFO data bus. These bits should drive the bus before asserting FIFO Write and can be read after tri-stating and asserting FIFO Read
RB1	I/O	S0_FIFO_D1	ISA FIFOs Data bus Bit 1	
RB2	I/O	S0_FIFO_D2	ISA FIFOs Data bus Bit 2	
RB3	I/O	S0_FIFO_D3	ISA FIFOs Data bus Bit 3	
RB4	I/O	S0_FIFO_D4	ISA FIFOs Data bus Bit 4	
RB5	I/O	S0_FIFO_D5	ISA FIFOs Data bus Bit 5	
RB6	I/O	S0_FIFO_D6	ISA FIFOs Data bus Bit 6	
RB7	I/O	S0_FIFO_D7	ISA FIFOs Data bus Bit 7	
RC0	O	S0_FIFO_RZ	FIFO (ISA Writes) Read (Assert Low)	Assert to read the In FIFO
RC1	O	S0_FIFO_WZ	PIC to ISA FIFO (ISA Reads) Write (Assert Low)	Assert to write the Out FIFO
RC2	O	S0_ISA_INT	PIC to ISA Interrupt Request (Assert ?)	Level Interrupt to ISA bus
RC3	I/O	S0_I2C_DATA	Wire Service Bus Clock (I2C)	Only used for I2C
RC4	I/O	S0_I2C_CLK	Wire Service Bus Data (I2C)	
RC5	O	S0_FIFO_RSTZ	In and Out FIFOs Reset (Assert Low)	Resets both FIFOs
RC6	O	S0_FIFO_IRTZ	In FIFO (ISA Writes) Retransmit (Assert Low)	Resets the Read pointer to 0
RC7	O	S0_FIFO_ORTZ	Out FIFO (ISA Reads) Retransmit (Assert Low)	Resets the Read pointer to 0
RD0	I/O	S0_CSR_D0	ISA External Data bus Bit 0 (Slave parallel port)	The slave parallel port is used as a bidirectional control and status register on the ISA bus.
RD1	I/O	S0_CSR_D1	ISA External Data bus Bit 1 (Slave parallel port)	
RD2	I/O	S0_CSR_D2	ISA External Data bus Bit 2 (Slave parallel port)	
RD3	I/O	S0_CSR_D3	ISA External Data bus Bit 3 (Slave parallel port)	
RD4	I/O	S0_CSR_D4	ISA External Data bus Bit 4 (Slave parallel port)	
RD5	I/O	S0_CSR_D5	ISA External Data bus Bit 5 (Slave parallel port)	
RD6	I/O	S0_CSR_D6	ISA External Data bus Bit 6 (Slave parallel port)	
RD7	I/O	S0_CSR_D7	ISA External Data bus Bit 7 (Slave parallel port)	
RE0	I	S0_CSR_RZ	ISA Read Slave Parallel Port (Assert Low)	Not directly manipulated after setting port D/E to act as slave parallel port
RE1	I	S0_CSR_WZ	ISA Write Slave Parallel Port (Assert Low)	

RE2	I	S0_CSR_SZ	ISA Slave Parallel Port Select (Assert Low)

Wire Service System Monitor A (System Type ID S1) Processor ID 3

Pin	Type	Name	Function	Notes
RA0	O	S1_FAN_HI	System Board fan speed to high (Assert Hi)	Assert on any SB fan failure
RA1	O	S1_SBFAN_LED	System Board fan fault LED	Assert on any SB fan failure
RA2	O	S1_BC_DS0	Bus/Core Speed Ratio and DIMM Select Mux Bit 0	During system reset these bits select bus/core speed ratio for all processors. Otherwise they select which DIMM presents its type on DIMM type port.
RA3	O	S1_BC_DS1	Bus/Core Speed Ratio and DIMM Select Mux Bit 1	
RA4	O	S1_BC_DS2	Bus/Core Speed Ratio and DIMM Select Mux Bit 2	
RA5	O	S1_BC_DS3	Bus/Core Speed Ratio and DIMM Select Mux Bit 3	
RB0	I/O	S1_LCD_D0	LCD Controller Data Bus bit 0	These lines make up the 8 bit data bus to the LCD display
RB1	I/O	S1_LCD_D1	LCD Controller Data Bus bit 1	
RB2	I/O	S1_LCD_D2	LCD Controller Data Bus bit 2	
RB3	I/O	S1_LCD_D3	LCD Controller Data Bus bit 3	
RB4	I/O	S1_LCD_D4	LCD Controller Data Bus bit 4	
RB5	I/O	S1_LCD_D5	LCD Controller Data Bus bit 5	
RB6	I/O	S1_LCD_D6	LCD Controller Data Bus bit 6	
RB7	I/O	S1_LCD_D7	LCD Controller Data Bus bit 7	
RC0	I	S1_FAN_TP	Tachometer pulse input from selected fan	Generally routed to counter
RC1	O?	S1_OK_TO_RUN	Drives SYS_PWRGOOD signal	System starts on 0->1 transition
RC2	I	S1_RESET_SW	Undebounced input from System Reset switch	
RC3	I/O	S1_I2C_DATA	Wire Service Bus Clock (I2C)	Only used for I2C
RC4	I/O	S1_I2C_CLK	Wire Service Bus Data (I2C)	
RC5	O	S1_FAN_SEL0	Fan Tachometer Multiplexer Select Bit 0	Used to select which fan tachometer pulse output is gated to S1_FAN_TP
RC6	O	S1_FAN_SEL1	Fan Tachometer Multiplexer Select Bit 1	
RC7	O	S1_FAN_SEL2	Fan Tachometer Multiplexer Select Bit 2	
RD0	I	S1_DIMM_D0	DIMM Type port bit 0	These lines make up an 8 bit port which on which the DIMM module in the slot selected by S1_BC_DS0..3 presents its type data if any. If no DIMM is present in the slot selected the DIMM type bits are all 1's.
RD1	I	S1_DIMM_D1	DIMM Type port bit 1	
RD2	I	S1_DIMM_D2	DIMM Type port bit 2	
RD3	I	S1_DIMM_D3	DIMM Type port bit 3	
RD4	I	S1_DIMM_D4	DIMM Type port bit 4	
RD5	I	S1_DIMM_D5	DIMM Type port bit 5	
RD6	I	S1_DIMM_D6	DIMM Type port bit 6	
RD7	I	S1_DIMM_D7	DIMM Type port bit 7	
RE0	O	S1_LCD_RS	LCD Controller Register Select	See LCD Controller data sheet for details of operation of these signals
RE1	O	S1_LCD_ENA	LCD Controller Register Enable	
RE2	O	S1_LCD_RW	LCD Controller Register Read/Write	

Wire Service System Monitor B (System Type ID S2) Processor ID 4

Pin	Type	Name	Function	Notes
RA0	O	S2_FLASH_LED	Display the Enable/Disable state of BIOS Flash ROM	Should track S2_FLASH_ENA
RA1	O	S2_SBFLT_LED0	System Board FRU LED Pin 0 (bicolor LED)	Drive in different combinations for OFF, AMBER, GREEN
RA2	O	S2_SBFLT_LED1	System Board FRU LED Pin 1 (bicolor LED)	
RA3	O	S2_OVRTMP_LED	Over Temperature LED	
RA4	I	S2_TEMP_CPU4	Thermal Fault - CPU 4	Indicator that CPU has exceeded temperature limit and faulted
RA5	I	S2_TEMP_CPU3	Thermal Fault - CPU 3	
RB0	O	S2_SB_JTAG	Enable System Board JTAG Chain TMS (Not Yet Implemented)	
RB1	O	S2_FLASH_WE	System BIOS FLASH Write Enable	
RB2	I	S2_FLASH_SW	System BIOS FLASH Write Enable Switch	
RB3	I	S2_NMI_SW	System Non-Maskable Interrupt (NMI) Switch	
RB4	I	S2_POK_CPU1	Power Good signal from CPU 1	Indicator that power regulator for CPU is operating correctly. Only valid if corresponding CPU is present (S2_PRES_CPUx)
RB5	I	S2_POK_CPU2	Power Good signal from CPU 2	
RB6	I	S2_POK_CPU3	Power Good signal from CPU 3	
RB7	I	S2_POK_CPU4	Power Good signal from CPU 4	
RC0	x		Unused	
RC1	x		Unused	
RC2	O	S2_NMI_CPU4	NMI Request for CPU 4	Toggle to cause NMI to CPU
RC3	I/O	S2_I2C_CLK	Wire Service Bus Data (I2C)	Only used for I2C
RC4	I/O	S2_I2C_CLK	Wire Service Bus Data (I2C)	
RC5	O	S2_NMI_CPU3	NMI Request for CPU 3	See S2_NMI_CPU4 above
RC6	O	S2_NMI_CPU2	NMI Request for CPU 2	
RC7	O	S2_NMI_CPU1	NMI Request for CPU 1	
RD0	I	S2_PRES_CPU1	Presence detection bit - CPU 1	Asserted when a processor inserted in the system board
RD1	I	S2_PRES_CPU2	Presence detection bit - CPU 2	
RD2	I	S2_PRES_CPU3	Presence detection bit - CPU 3	
RD3	I	S2_PRES_CPU4	Presence detection bit - CPU 4	
RD4	I	S2_ERROR_CPU1	Processor Fault bit - CPU 1	Processor either failed BIST on startup or later other fault. Only valid if corresponding CPU is present (S2_PRES_CPUx)
RD5	I	S2_ERROR_CPU2	Processor Fault bit - CPU 2	
RD6	I	S2_ERROR_CPU3	Processor Fault bit - CPU 3	
RD7	I	S2_ERROR_CPU4	Processor Fault bit - CPU 4	
RE0	O	S2_SYSFLT_LED	System Fault summary LED	
RE1	I	S2_TEMP_CPU2	Thermal Fault - CPU 2	See S2_TEMP_CPU4 above
RE2	I	S2_TEMP_CPU1	Thermal Fault - CPU 1	

Wire Service System Recorder (System Type ID S3) Processor ID 1

Pin	Type	Name	Function	Notes
RA0	O	S3_NVRAM_A8	NVRAM Address Bit 8	NVRAM Address Bus Bits 8-13
RA1	O	S3_NVRAM_A9	NVRAM Address Bit 9	
RA2	O	S3_NVRAM_A10	NVRAM Address Bit 10	
RA3	O	S3_NVRAM_A11	NVRAM Address Bit 11	
RA4	O	S3_NVRAM_A12	NVRAM Address Bit 12	
RA5	O	S3_NVRAM_A13	NVRAM Address Bit 13	
RB0	I/O	S3_NVRAM_D0	NVRAM Data Bit 0	NVRAM 8 Bit Data Bus
RB1	I/O	S3_NVRAM_D1	NVRAM Data Bit 1	
RB2	I/O	S3_NVRAM_D2	NVRAM Data Bit 2	
RB3	I/O	S3_NVRAM_D3	NVRAM Data Bit 3	
RB4	I/O	S3_NVRAM_D4	NVRAM Data Bit 4	
RB5	I/O	S3_NVRAM_D5	NVRAM Data Bit 5	
RB6	I/O	S3_NVRAM_D6	NVRAM Data Bit 6	
RB7	I/O	S3_NVRAM_D7	NVRAM Data Bit 7	
RC0	O	S3_NVRAM_CSZ	NVRAM Chip Select (Negative Logic)	Control signals for NVRAM - See Dallas DS1245 data sheet
RC1	O	S3_NVRAM_OEZ	NVRAM Output Enable (Negative Logic)	
RC2	O	S3_NVRAM_WEZ	NVRAM Write Enable (Negative Logic)	
RC3	I/O	S3_I2C_CLK	Wire Service Bus Data (I2C)	Only used for I2C
RC4	I/O	S3_I2C_CLK	Wire Service Bus Data (I2C)	
RC5	O	S3_NVRAM_A14	NVRAM Address Bit 14	NVRAM Address Bus Bits 14-16
RC6	O	S3_NVRAM_A15	NVRAM Address Bit 15	
RC7	O	S3_NVRAM_A16	NVRAM Address Bit 16	
RD0	O	S3_NVRAM_A0	NVRAM Address Bit 0	NVRAM Address Bus Bits 0-7
RD1	O	S3_NVRAM_A1	NVRAM Address Bit 1	
RD2	O	S3_NVRAM_A2	NVRAM Address Bit 2	
RD3	O	S3_NVRAM_A3	NVRAM Address Bit 3	
RD4	O	S3_NVRAM_A4	NVRAM Address Bit 4	
RD5	O	S3_NVRAM_A5	NVRAM Address Bit 5	
RD6	O	S3_NVRAM_A6	NVRAM Address Bit 6	
RD7	O	S3_NVRAM_A7	NVRAM Address Bit 7	
RE0	O	S3_RTC_CLK	Real Time Clock - Data Clock	See Dallas DS1603 data sheet
RE1	I	S3_RTC_DATA	Real Time Clock - Serial Data	
RE2	O	S3_RTC_RSTZ	Real Time Clock - Protocol Reset (Negative Logic)	

Wire Service Backplane (System Type ID S4) Processor ID 2

Pin	Type	Name	Function	Notes
RA0	A	S4_VOLTS_P5V	Analog measure of system +5 volt main supply	Use D/A converter to read voltages as 0-255. Calibration constants are determined externally
RA1	A	S4_VOLTS_P3V	Analog measure of system +3.3 volt main supply	
RA2	A	S4_VOLTS_P12V	Analog measure of system +12 volt main supply	
RA3	A	S4_VREF	Voltage Reference for A/D converter	Unused
RA4	x			
RA5	A	S4_VOLTS_N12V	Analog measure of system -12 volt main supply	See S4_VOLTS_P5V
RB0	I/O	S4_PSN_CAN1	Presence and Serial Number I/O for Canister 1	These are all lines to one wire serial data EPROMS. See Dallas DS250x data sheet for programming information
RB1	I/O	S4_PSN_CAN2	Presence and Serial Number I/O for Canister 2	
RB2	I/O	S4_PSN_CAN3	Presence and Serial Number I/O for Canister 3	
RB3	I/O	S4_PSN_CAN4	Presence and Serial Number I/O for Canister 4	
RB4	I/O	S4_PSN_CAN5	Presence and Serial Number I/O for Canister 5	
RB5	I/O	S4_PSN_CAN6	Presence and Serial Number I/O for Canister 6	
RB6	I/O	S4_PSN_CAN7	Presence and Serial Number I/O for Canister 7	
RB7	I/O	S4_PSN_CAN8	Presence and Serial Number I/O for Canister 8	
RC0	x			
RC1	I	S4_ACOK_PS3	A/C Input OK to Power Supply 3	Should check only if PSN for power supply indicates presence.
RC2	I	S4_ACOK_PS2	A/C Input OK to Power Supply 2	
RC3	I/O	S4_I2C_CLK	Wire Service Bus Data (I2C)	Only used for I2C
RC4	I/O	S4_I2C_CLK	Wire Service Bus Data (I2C)	
RC5	I	S4_ACOK_PS1	A/C Input OK to Power Supply 1	See S4_ACOK_PS3
RC6	O	S4_POWER_ON	Enable main output from power supplies	
RC7	I	S4_POWER_SW	Power On/Off switch (undebounced)	
RD0	I/O	S4_PSN_PS1	Presence and Serial Number for Power Supply 1	These are all lines to one wire serial data EPROMS. See Dallas DS250x data sheet
RD1	I/O	S4_PSN_PS2	Presence and Serial Number for Power Supply 2	
RD2	I/O	S4_PSN_PS3	Presence and Serial Number for Power Supply 3	
RD3	I/O	S4_PSN_BP	Presence and Serial Number for Backplane	Dallas DS250x also
RD4	I/O	S4_PSN_SB	Presence and Serial Number for System Board	Dallas DS250x also
RD5	I	S4_BP_TYPE	Backplane Type (0: Small 1: Large)	
RD6	I/O	S4_TEMP_SCL	Temperature Bus Clock	I2C local bus for temperature probes at different system points
RD7	I/O	S4_TEMP_SDA	Temperature Bus Serial Data	
RE0	I	S4_DCOK_PS3	D/C Output OK from Power Supply 3	Should check only if PSN for power supply indicates presence.
RE1	I	S4_DCOK_PS2	D/C Output OK from Power Supply 2	
RE2	I	S4_DCOK_PS1	D/C Output OK from Power Supply 1	

Wire Service Canister (System Type ID S5) Processor ID 2x where x is the slot ID

Pin	Type	Name	Function	Notes
RA0	O	S5_P12V_ENA	Turns on +/- 12 volt to all PCI slots	Used to sequence power to PCI cards.
RA1	O	S5_P5V_ENA4	Turns on +5 volts to PCI slot 4	
RA2	O	S5_P5V_ENA3	Turns on +5 volts to PCI slot 3	
RA3	O	S5_P5V_ENA2	Turns on +5 volts to PCI slot 2	
RA4	O	S5_P5V_ENA1	Turns on +5 volts to PCI slot 1	
RA5	O	S5_PSLOT5_ENA	Turns on power on slot 5 card (if any)	Always turn on last
RB0	I	S5_CAN_A0	Canister Address bit 0	Determine the Wire Service bus address of this canister
RB1	I	S5_CAN_A1	Canister Address bit 1	
RB2	I	S5_CAN_A2	Canister Address bit 2	
RB3	I	S5_PRSNT_S5	Special Slot 5 (IOP/PCI jumper) present	Indicates something is in slot 5
RB4	I/O	S5_PSN_S5	Present Serial Number for special slot 5	From DS 250x in slot 5 card (if IOP)
RB5	x			
RB6	x			
RB7	x			
RC0	I	S5_FAN_TP	Tachometer pulse input from selected fan	Generally routed to counter
RC1	O	S5_FAN_SEL0	Fan Tachometer Multiplexer Select Bit 0	Select which fan to monitor tach.
RC2	x			
RC3	I/O	S5_I2C_CLK	Wire Service Bus Data (I2C)	Only used for I2C
RC4	I/O	S5_I2C_CLK	Wire Service Bus Data (I2C)	
RC5	O	S5_CANFAN_LED	Canister fan fault LED	Assert on any Canister fan failure
RC6	O	S5_CANFLT_LED0	Canister FRU LED Pin 0 (bicolor LED)	Drive in different combinations for OFF, AMBER, GREEN
RC7	O	S5_CANFLT_LED1	Canister FRU LED Pin 1 (bicolor LED)	
RD0	I	S5_PRSNT_S1A	PCI card present in Slot 1 (A pin)	PCI slots have 2 presence pins - see PCI spec for usage and meaning.
RD1	I	S5_PRSNT_S1B	PCI card present in Slot 1 (B pin)	
RD2	I	S5_PRSNT_S2A	PCI card present in Slot 2 (A pin)	
RD3	I	S5_PRSNT_S2B	PCI card present in Slot 2 (B pin)	
RD4	I	S5_PRSNT_S3A	PCI card present in Slot 3 (A pin)	
RD5	I	S5_PRSNT_S3B	PCI card present in Slot 3 (B pin)	
RD6	I	S5_PRSNT_S4A	PCI card present in Slot 4 (A pin)	
RD7	I	S5_PRSNT_S4B	PCI card present in Slot 4 (B pin)	
RE0	O	S5_CAN_JTAG	Enable Canister Board JTAG Chain TMS	Required to select the JTAG chain
RE1	O	S5_NMI_S5	NMI card is special slot 5 (IOP)	Toggle to NMI IOP in slot 5
RE2	O	S2_FAN_HI	Canister fan speed to high (Assert Hi)	Assert on any Canister fan failure

Wire Service Remote Interface (System Type ID S6) Processor ID 11

Pin	Type	Name	Function	Notes
RA0	I/O	S6_PSN_RI	Serial Number information for Remote Interface	
RA1	x			
RA2	x			
RA3	x			
RA4	x			
RA5	x			
RB0	O	S6_MODEM_DTR	Modem Signal (Data Terminal Ready)	
RB1	I	S6_MODEM_DSR	Modem Signal (Data Set Ready)	
RB2	I	S6_MODEM_CD	Modem Signal (Carrier Detect)	
RB3	I	S6_MODEM_RI	Modem Signal (Ring Indicate)	
RB4	O	S6_MODEM_RTS	Modem Signal (Request To Send)	
RB5	I	S6_MODEM_CTS	Modem Signal (Clear To Send)	
RB6	x			
RB7	x			
RC0	x			
RC1	x			
RC2	x			
RC3	I/O	S6_I2C_CLK	Wire Service Bus Data (I2C)	Only used for I2C
RC4	I/O	S6_I2C_CLK	Wire Service Bus Data (I2C)	
RC5	x			
RC6	O	S6_MODEM_TXD	Modem Signal (Transmit Data)	Controlled by chip serial interface
RC7	I	S6_MODEM_RXD	Modem Signal (Receive Data)	
RD0	I/O	S3_SRAM_D0	SRAM Data Bit 0	
RD1	I/O	S3_SRAM_D1	SRAM Data Bit 1	
RD2	I/O	S3_SRAM_D2	SRAM Data Bit 2	
RD3	I/O	S3_SRAM_D3	SRAM Data Bit 3	
RD4	I/O	S3_SRAM_D4	SRAM Data Bit 4	
RD5	I/O	S3_SRAM_D5	SRAM Data Bit 5	
RD6	I/O	S3_SRAM_D6	SRAM Data Bit 6	
RD7	I/O	S3_SRAM_D7	SRAM Data Bit 7	
RE0	x			
RE1	x			
RE2	x			

Wire Service Network Memory Map

This section defines the Wire Service Network Memory Map for the first Raptor system. Its purpose is to identify all Wire Service addressable entities and describe their function and any special information about them.

In the following table:
Name is the official symbolic name for the Wire Service entity.
Type is the data type for the entity from the data type described previously.
PID is the processor ID (hexadecimal) of the Wire Service Processor where the data resides.
Note: Canister Wire Service PID is determined by adding 0x20 to the canister addresss on S5_CAN_A[0-2].
Description is a short narrative on the function of the entity.
Notes describe any other special information or actions involving the entity

Name	Type	PID	Description	Notes
WS_DESC_Pn	STRING	Any	Wire Service Processor Type/Description	
WS_REV_Pn	STRING	Any	Wire Service Software Revision/Date Info	
WS_SB_FAN_HI	BIT	03	System Board Fans HI	Controls S1_FAN_HI. Set on 0->1 transition of WS_SB_FAN_LED. Cleared by other software
WS_SB_FAN_LED	BIT	03	System Board Fan Fault LED	Controls S1_SBFAN_LED. It is set whenever any WS_SB_FANFAULT is non-zero. Log 0->1 transition
WS_SB_BUSCORE	BYTE	03	System Board BUS/CORE speed ratio to use on reset	Value is asserted on S1_BC_DS[0-3] unless reading DIMM types. Set to 0 on power on.
WS_SYS_LCD1	STRING	03	Value to display on LCD line 1	For a Nx2 display wher LCD1 is the top line and LCD2 is the bottom line. No wrapping occurs from one line to the other. Manipulates S1_LCD_D[0-7], S1_LCD_RS, S1_LCD_ENA, S1_LCD_RW
WS_SYS_LCD2	STRING	03	Value to display on LCD line 2	
WS_SB_FAN_DATA	STRING	03	System Board Fan speed data in fan number order	Approximately every second a fan is selected by S1_FAN_SEL[0-2] and monitored via S1_FAN_TP driving a counter for a known period of time. The counter is then loaded into the appropriate fan speed. If WS_SB_FAN_HI is not set then the speed is compared against WS_SB_FAN_LOLIM. If fan is slow set

				appropriate bit in WS_SB_FANFAULT otherwise clear it
WS_SB_FANFAULT	BYTE	03	System Board Fan fault bits	Bit 0 = Fan1, bit 1 = Fan2, etc. A non-zero byte value implies at least one fan faulted.
WS_SB_FAN_LOLIM	BYTE	03	Fan speed low speed fault limit	Set to ??? on power on
WS_SB_DIMM_TYPE	STRING	03	The type of DIMM in each DIMM socket as a 16 byte string	When read asserts value of 0..15 on S1_BC_DS[0-3] and then returns value of S1_DIMM_D[0-7] for each value.
WS_SB_FLASH_ENA	BIT	04	Indicates FLASH ROW write enabled	Set/Cleared by debounced 0->1 transition of S2FLASH_SW. Controls state of S2_FLASH_WE and S2_FLASH_LED.
WS_SB_FRU_FAULT	BIT	04	Indicates the FRU status	At power on starts at 1. Controls S2_SBFLT_LED[0-1] for bicolor LED colors 0=Green 1=Amber, Cleared by other software
WS_SYS_OVERTEMP	BIT	04	Indicates Overtemp fault	At power on is set. Controls S2_OVRTMP_LED. Controlled by wire service backplane processor.
WS_SB_JTAG	BIT	04	Enables JTAG chain on system board	Clear at power on. Controls S2_SB_JTAG
WS_SB_CPU_PRES	BYTE	04	CPU Presence bits (LSB = CPU1)	Assemble from S2_PRES_CPU[1-4]
WS_SB_CPU_ERR	BYTE	04	CPU Error bits (LSB = CPU1)	Assemble from S2_ERROR_CPU[1-4]
WS_SB_CPU_TEMP	BYTE	04	CPU Thermal fault bits (LSB = CPU1)	Assemble from S2_TEMP_CPU[1-4]
WS_SB_CPU_POK	BYTE	04	CPU Power OK (LSB = CPU1)	Assemble from S2_POK_CPU[1-4]
WS_NMI_MASK	BYTE	04	CPU NMI processor mask (LSB=CPU1)	Defaults to all ones on power up
WS_NMI_REQ	BIT	04	NMI Request bit	When set pulse S2_NMI_CPUn corresponding to each bit set in WS_NMI_MASK. Then clear request bit. Log Action
WS_SYSFAULT	BIT	04	System Fault Summary	This bit is set if any faults detected in the system. Controls S2_SYSFLT_LED Bits scanned WS_SP_CPU_FAULT, WS_SB_FRU_FAULT (other faults?)
WS_SB_CPU_FAULT	BIT	04	CPU Fault Summary	This bit is set if ((WS_SB_CPU_ERR | WS_SB_CPU_TEMP | ~WS_SB_CPU_POK) & ~WS_SB_CPU_PRES) != 0. Log 0->1

				transition with CPU bytes.
WS_BP_P5V	BYTE	02	Analog Measure of +5 volt main supply	read from S4_VOLTS_P5v
WS_BP_P3V	BYTE	02	Analog Measure of +3.3 volt main supply	read from S4_VOLTS_P3v
WS_BP_P12V	BYTE	02	Analog Measure of +12 volt main supply	read from S4_VOLTS_P12v
WS_BP_P5V	BYTE	02	Analog Measure of -12 volt main supply	read from S4_VOLTS_N12V
WS_SYS_CAN_PRES	BYTE	02	Presence bits for canisters (LSB=1, MSB=8)	controlled by S4_PSN_CAN[1-8]. A previous value byte needs to be maintained so canister transitions can be recognized. Previous value initialized to zero. Periodic monitor scans for new canisters. When new canister is recognized read full serial data and store in WS_SYS_CAN_SERIALn then log and send event
WS_SYS_PS_PRES	BYTE	02	Presence bits for power supplies (LSB=1, MSB=3)	controlled by S4_PSN_PS[1-3]. A previous value byte needs to be maintained so power supply transitions can be recognized. Previous value initialized to zero. Periodic monitor scans for new power supplies. When new power supply is recognized read full serial data and store in WS_SYS_PS_SERIALn then log and send event
WS_SYS_PS_ACOK	BYTE	02	Power supply ACOK status (LSB=1, MSB=3)	controlled by S4_ACOK_PS[1-3]. A previous value byte needs to be maintained so power supply transitions can be recognized. Previous value initialized to zero. Periodic monitor scans for changes is ACOK and sends events
WS_SYS_PS_DCOK	BYTE	02	Power supply DCOK status (LSB=1, MSB=3)	controlled by S4_DCOK_PS[1-3]. A previous value byte needs to be maintained so power supply transitions can be recognized. Previous value initialized to zero. Periodic monitor scans for changes is ACOK and sends events
WS_SYS_BP_TYPE	BYTE	02	Type of system backplane currently only two types Type 0= 4 canister (small) and Type 1= 8 canister (large)	controlled by S4_BP_TYPE
WS_SYS_TEMP_DATA	STRING	02	Temperatures of all sensors on temperature bus in address order	controlled by reading Dallas temperature transducers connected to serial bus on S4_TEMP_SDA and S4_TEMP_SCL

WS_SYS_TEMP_WARN	BYTE	02	Warning temperature. Initialized to ???	If any WS_SYS_TEMP_xxn exceeds this value, log, send event, set WS_SYS_OVERTEMP
WS_SYS_TEMP_SHUT	BYTE	02	Shutdown temperature. Initialized to ???	If any WS_SYS_TEMP_xxn exceeds this value, log and clear WS_SYS_POWER
WS_SYS_REQ_POWER	BIT	02	Set to request main power on	
WS_SYS_POWER	BIT	02	Controls system master power S4_POWER_ON	When this bit is set 0->1 set S4_POWER_ON, WS_SYS_RUN = 0, WS_SYS_RSTIMER = 5 and log, When this bit is cleared clear S4_POWER_ON and log.
WS_SYS_RSTIMER	BYTE	02	Used to delay reset/run until power stabilized	Counts down to 0 at 10 counts per second. When 1->0 transition sets WS_SYS_RUN.
WS_SYS_WDOG	BYTE	02	System watchdog timer	Counts down to 0 at 10 countsper secong When 1->0 transition takes action(s) specified by WS_WDOG_RESET and WS_WDOG_CALLOUT.
WS_SYS_RUN	BIT	02	Controls the system halt/run line S1_OK_TO_RUN.	If this bit is cleared, clear S1_OK_TO_RUN and log. If this bit is set, set S1_OK_TO_RUN and log.
WS_CAN_POWER	BIT	2x	Controls canister PCI slot power	When set then set S5_P5V_ENA[1..4],S5_P12V_ENA in that order with small (about 1 ms) delay between each, then log. When cleared then clear S5_P12V_ENA, S5_P5V_ENA[1..4] then log
WS_CAN_PCI_PRESENT	BYTE	2x	Reflects PCI card slot[1..4] presence indicator pins (MSB to LSB) 4B,4A,3B,3A,2B,2A,1B ,1A	Reflects data from S5_PRSNT_S[1..4][A/B]
WS_CAN_S5_PRESENT	BIT	2x	Indicates the presence of something in slot 5	Reflects S5_PRSNT_S5
WS_CAN_S5_SMART	BIT	2x	Indicates something other than a passive board in slot 5	On power up attempt to read Dallas serial number chip using S5_PSN_S5. If present set this bit and read full serial data and store in WS_SYS_CAN_IOP_SERIALn
WS_CAN_FAN_HI	BIT	2x	Canister Fans HI	Controls S1_FAN_HI. Set on 0->1 transition of WS_SB_FAN_LED. Cleared by other software
WS_CAN_FAN_LED	BIT	2x	Canister Fan Fault	Controls S5_CANFAN_LED. It is set whenever any WS_CAN_FANFAULTn is

			LED	set. Log 0->1 transition
WS_CAN_FANFAULT	BYTE	03	Canister Fan Fault Bits	
WS_CAN_FAN_DATA	STRING	2x	Canister Fan speed data	Approximately every second a fan is selected by S5_FAN_SEL0 and monitored via S5_FAN_TP driving a counter for a known period of time. The counter is then loaded into the appropriate fan speed. If WS_CAN_FAN_HI is not set then the speed is compared against WS_CAN_FAN_LOLIM. If fan is slow set appropriate WS_CAN_FANFAULTn otherwise clear it
WS_CAN_FAN_LOLIM	BYTE	2x	Fan low speed fault limit	Set to equivalent of xxx RPM on power on
WS_CAN_JTAG_ENA	BIT	2x	Enable JTAG TMS chain for canister	Copy set value to S5_CAN_JTAG
WS_CAN_NMI_S5	BIT	2x	NMI card in slot 5	when set, pulse S2_NMI_S5
WS_RI_CD	BIT	11	Status of Remote Port Modem CD	Follows S6_MODEM_CD
WS_RI_DTR	BIT	11	State of Remote Port Modem DTR	Controls S6_MODEM_DTR
WS_RI_DSR	BIT	11	Status of Remote Port Modem DSR	Follows S6_MODEM_DSR
WS_RI_RTS	BIT	11	Status of Remote Port Modem RTS	Controls S6_MODEM_RTS
WS_RI_CTS	BIT	11	Status of Remote Port Modem CTS	Follows S6_MODEM_CTS
WS_RI_CALLOUT	BYTE	11	Controls Call out Script activation	If written to it initiates Call out sequence programmed in WS_SYS_CALL_SCRIPT passing value as argument to script. Log it (Format of Script Programs TBD)
WS_RI_EVENTS	EVENT	11	Remote Interface Event BitVecctor	See Event Data type description in prior section.
WS_SI_EVENTS	EVENT	10	System Interface Event BitVector	See Event Data type description in prior section.
WS_SYS_LOG	LOG	01	System Log	The system log kept in NVRAM (See LOG data type in previous section)
WS_SYS_SCREEN	SCREEN	01	System Screen	A copy of the most recent character mode screen from the system video display (See SCREEN data type in previous section)

WS_SYS_BOOTFLAG1	BYTE	01	System Boot Flag 1	Used by BIOS, Diagnostics and Operating systems to control behavior. Values defined by those entities.
WS_SYS_BOOTFLAG2	BYTE	01	System Boot Flag 2	
WS_SYS_BOOTFLAG3	BYTE	01	System Boot Flag 3	
WS_SYS_BOOTFLAG4	BYTE	01	System Boot Flag 4	
WS_SYS_BOOTDEVS	STRING	01	System Boot Devices	Used by BIOS to determine boot device
WS_SYS_SB_SERIAL	STRING	01	Last known System Board serial data	
WS_SYS_BP_SERIAL	STRING	01	Last known Back Plane serial data	
WS_SYS_RI_SERIAL	STRING	01	Last known Remote Interface serial data	
WS_SYS_CAN_SERIAL[1-8]	STRING	01	Last known Canister [1-8] Serial data	May be zero length if no canister ever seen
WS_SYS_IOP_SERIAL[1-8]	STRING	01	Last known IOP in Canister [1-8] Serial data	May be zero length if no canister ever seen or current canister has no IOP
WS_PASSWORD	STRING	01	The access password for Wire Service	Used only for remote access
WS_WDOG_RESET	BIT	01	This is a bit controlling system on a wathcdog timeout.	This bit is queried on a watch dog timeout. If it is 1 a system reset takes place.
WS_WDOG_CALLOUT	BIT	01	This is a bit controlling callout on a wathcdog timeout.	This bit isqueried on a watch dog timeout. If it is a 1 a callout is initialted.
WS_POWERUP_HOLD	BIT	01	The mode controls action of system on A/C power available	If this bit is clear, when A/C power first becomes available the system will power on. If bit is set the system will not powerup automatically.
WS_CALLOUT_SCRIPT	STRING	01	The callout script for remote notification	Used for remote access
WS_SI_QUEUE	QUEUE	01	Queue of data going to System Interface	See Queue data type in previous section
WS_RI_QUEUE	QUEUE	01	Queue of data going to Remote Interface	See Queue data type in previous section
WS_SYS_XDATA	BYTE ARRAY	01	Byte Array for storage of arbitrary external data in NVRAM	Wire Service just maintains this data area and is unaware of the meaning of any data stored in it.
WS_SYS_XDATA_KBYTES	BYTE	01	Size of the WS_SYS_XDATA in kilobytes	Necessary for memory management of the data area

WS_NVRAM_RESET	BYTE	01	Trigger to reset NVRAM Data	When the value 0x5a is written to this variable, all of NVRAM is reinitialized or cleared. This byte should be that last cleared to indicate the reinitialization is complete.

Wire Services Processor Functions/Requirements

The previous section either states or directly implies many monitoring, control and feedback operations carried out by the different wire service processors. In addition the following functions or actions must be implemented:

- Monitor and debounce all switches. All switches are considered to be momentary contact type and all actions are taken on the 0 to 1 transition. All 0 to 1 transitions are logged.

  S4_POWER_SW -Toggle the state of WS_SYS_POWER.

  S1_RESET_SW - If WS_SYS_RUN is set, clear WS_SYS_RUN. Next if WS_SYS_POWER is set, set WS_SYS_RSTIMER to 5.

  S2_FLASH_SW - Toggle the state of WS_SB_FLASH_ENA.

  S2_NMI_SW - Set WS_NMI_REQ.

- Startup process when backplane Wire Service processors are first powered up:
  1) If S4_POWER_SW is depressed when starting and for at least 5 seconds after, this causes Wire Service backplane processor to reinitialize the contents of the NVRAM by writing a special value of 0x5a to WS_NVRAM_RESET and then reading WS_NVRAM_RESET until it returns 0.
  2) Read system board serial information on S4_PSN_SB and store it in WS_SYS_SB_SERIAL and backplane board serial on S4_PSN_BP and store it in WS_SYS_BP_SERIAL using the Dallas one wire serial protocol..
  3) If WS_SYS_PS_ACOK is not zero, then check WS_POWERUP_HOLD.
  4) If WS_POWERUP_HOLD is clear then set WS_SYS_POWER.

- Power on process when WS_SYS_POWER is set:
  1) Log power on requested.
  2) Log WS_SYS_TEMP_SHUT, WS_SYS_TEMP_WARN, WS_SYS_TEMP_DATA.
  3) If WS_SYS_OVERTEMP is not set, continue with next step otherwise stop.
  4) Set S4_POWER_ON.
  5) Set WS_SYS_RSTIMER to 5 (5/10 second) and stop.

- Power off process when WS_SYS_POWER is cleared:
  1) Log power off request.
  2) Clear WS_SYS_RUN and WS_SYS_RSTIMER.
  3) Clear S4_POWER_ON.

- Reset sequence when WS_SYS_RSTIMER counts down from 1 to 0:
  1) If Log reset requested.
  2) Log WS_SYS_PS_PRES, WS_SYS_PS_ACOK, WS_SYS_PS_DCOK, WS_BP_P5V,

WS_BP_P3V, WS_BP_P12V, WS_BP_N12V.
3) If WS_SYS_PS_DCOK not 0, continue with next step otherwise stop.
4) Set WS_SYS_RUN.

- Temperature monitor process every several seconds:
  1) Reads all temperature sensor data into WS_SYS_TEMP_DATA
  2) Compares each WS_SYS_TEMP_DATA item to WS_SYS_TEMP_SHUT. If any data temperature exceeds limit and WS_SYS_POWER is set, clear WS_SYS_POWER, log WS_SYS_TEMP_SHUT, WS_SYS_TEMP_WARN, WS_SYS_TEMP_DATA and stop.
  3) Compares each WS_SYS_TEMP_DATA item to WS_SYS_TEMP_WARN. If any data temperature exceeds limit and WS_SYS_OVERTEMP is clear, set WS_SYS_OVERTEMP, send a TEMPERATURE event to the system and remote interfaces, log WS_SYS_TEMP_SHUT, WS_SYS_TEMP_WARN, WS_SYS_TEMP_DATA and stop.
  4) Otherwise if no limits are exceeded and WS_SYS_OVERTEMP is set, clear WS_SYS_OVERTEMP and send a TEMPERATURE event to the system and remote interfaces.

- Monitor fault conditions for system fault and control WS_SYSFAULT_LED. (Needs further work on mechanism)

- Several times per second the backplane Wire Service processor monitors S4_PSN_CANn for changes in presence of the canisters. Each S4_PSN_xxx line is really a Dallas one wire serial bus signal attached to a Dallas serial number chip. To detect the presence of a canister, a reset pulse must be sent by the Wire Service processor and then the canister's presence pulse must be detected. Unless there is a change in the presence of a canister nothing more needs to be done. If a change in presence is detected, the WS_SYS_CAN_PRES data must be updated, a CANISTER event sent to the system and remote interface processors and the old and new WS_SYS_CAN_PRES data is logged. If the a canister was removed, nothing more needs to be done. If a canister was inserted, then the full serial data must ber read from the canister using the Dallas one wire protocol and stored in the appropriate WS_SYS_CAN_SERIALx string.

- Several times per second the backplane Wire Service processor monitors S4_PSN_PSn for changes in presence of the power supplies. Each S4_PSN_xxx line is really a Dallas one wire serial bus signal attached to a Dallas serial number chip. To detect the presence of a power supply, a reset pulse must be sent by the Wire Service processor and then the canister's presence pulse must be detected. Unless there is a change in the presence of a power supply nothing more needs to be done. If a change in presence is detected, the WS_SYS_PS_PRES data must be updated, a POWER_SUPPLY event sent to the system and remote interface processors and the old and new WS_SYS_PS_PRES data is logged. If the a power supply was removed, nothing more needs to be done. If a power supply was inserted, then the full serial data must ber read from the power supply using the Dallas one wire protocol and stored in the appropriate WS_SYS_PS_SERIALx string.

- Once per second, all Wire Service processors that have fan monitoring inputs collect data from the next fan in sequence to be monitored and store the fan spped data in the appropriate slot in

WS_xxx_FAN_DATA for that fan. If the value for the fan speed data is below WS_xxx_FAN_LOWLIM, the appropriate bit in WS_xxx_FAN_FAULT is set , the Wire Service processer ID, WS_xxx_FAN_LOWLIM, and WS_xxx_FAN_DATA are logged and WS_xxx_FAN_HI is set. If this occurs, the only way to reset the fan fault LED and fan speed is by a system or remote reset of WS_xxx_FAN_FAULT and WS_xxx_FAN_HI or power cycling the appropriate board.

- The Wire Service remote interface processor on power up reads serial number data from S6_PSN_RI using the Dallas one wire serial protocol and stores it in WS_SYS_RI_SERIAL.

- Events are always sent to both WS_SI_EVENTS and WS_RI_EVENTS queues with no retry if no processor response from target.

Wire Service and Raptor BIOS Interactions

The following are items which it is known that the BIOS must implement.

- BIOS must determine current DIMM configuration by reading WS_SB_DIMM_TYPE to determine DIMM types and then validating that this is an OK configuration. Also, memory can be initially sized this way.

- BIOS can read WS_SB_CPU_PRES To determine which processors are present and then go out on the bus and somehow get the processor type and speed information to determine the correct BUS/CORE speed ratio. Once that is determined. The BIOS reads WS_SB_BUSCORE to determine the actual BUS/CORE speed ratio and if it is incorrect, write the correct one to WS_SB_BUSCORE, set WS_SYS_RSTIMER to 5 (5/10 second), and clear WS_SYS_RUN. This will reset the system using the new BUS/CORE ratio. (Note: If WS_SYS_RSTIMER is not set, wire service will not set WS_SYS_RUN after the BIOS clears it and the system will remain halted.

- The BIOS must read WS_SYS_CAN_PRES to determine which canisters are present and then set WS_CAN_POWER on each canister to enable power to the PCI cards before configuring the PCI busses.

- All WS_xx_FRU_LEDs are amber at power up. The BIOS sets WS_xx_FRU_LED to green when satisfied that the system board or canister is ok. Diagnostics may turn WS_xx_FRU_LEDs to amber on diagnostic failure fault.

- The BIOS, when remote control is activated, should update the Wire Service video memory image from either the true character mode video memory or as it updates true video memory. The BIOS under the same conditions should monitor the system interface for events indicating arrival of keycodes in the system queue and then process those codes as normal keystrokes. This same mechanism can also be used for floppy/hard disk emulation. (Note: how remote control is activated is not yet specified)

Wire Service Issues Needing Resolution

1) Does Fan Fault roll up into system fault LED?

4) If there is a CPU thermal fault should the over temp LED turn on or system board fault LED?

5) How much monitoring/logging should be done for CPU fault signals?

6) What are all the conditions that turn on system fault summary LED?

7) Are there any times that the system fault summary must be valid? May not be valid? Response times?

Wire Service Remote Interface Serial Protocol

The Wire Service Remote Serial protocol is used to communicate Wire Service messages across a serial link from a Wire Service Remote Interface processor attached to a Raptor to a Wire Service Remote management processor. It encapsulates Wire Services messages in a transmission envelope to provide error free communications and link security.

Wire Service Call out Script Syntax Definition

To be specified

Document Revision History

Version 0.9 - Initial Version
Version 1.0 - Added state diagram to system interface description
- Completed Network Memory Map (except for precise address specification and debug)
- Added section detailing internal Wire Service monitoring and processes.
- Added section for BIOS required processing
- Added section for unresolved issues
- Added this revision history section

Version 1.1 - Converted to Little Endian (Intel) byte order for integers and address larger than one byte
- Removed requirement for system processors to know about check bytes on messages
- Changed system interface allocation mechanism slightly
- Added more description and conventions for log entries
- Changed several Wire Service memory definitions for easier use (fan data and DIMMs)
- Added several new Wire Service memory definitions from review and to support dial out
- Added watchdog to Wire Service memory
- Cleand up and expanded section on Wire Service Processing Functions
- Added Wire Service hardware diagram
- Added NVRAM reset mechanism
- Added preliminary BIOS remote control description

Version 1.2 - Changed definition and semantics of event data type from queue to bit vector
- Split LCD display into two lines
- Added WS_SYS_BOOTDEVS string for describing various boot device selection information for BIOS

EXHIBIT B

Raptor System

A Bird's Eye View

Karl Johnson (KJ)

Revision 0.99

November 2, 1995

Control Diagnostic and Monitor Subsystem

The control of Raptor is completly "Fly By Wire" - i.e. no physical switch directly controls any function and no indicator is directly controlled by system hardware. All such functions, refered to as Out Of Band functions[1], are controlled through a Control Diagnostic and Monitor (CDM) subsystem implemented by small distributed CDM processors connected on a 400 kbs I^2C serial bus (the CDM Bus). Critical CDM components are powered by the bias power from any of the power supplies, which means that CDM basic CDM functions are avaiable so long as A/C power is available at the input to any of the power supplies. The CDM subsystem supervises or monitors the following system features:

[1]See Glossary

- Power supplies - Presence, status, A/C good, Power On/Off and Output voltages.
- Environment - Ambient and exhaust temperatures, Fan speed, speed control, Fan fault and Overtemp indicators.
- Processor - CPU Presence, Power OK, Overtemp and Fault, NMI control, System reset, Memory type/location and Bus/Core speed ratio.
- I/O - I/O Canister insertion/removal and status indicator, PCI card presence, PCI card power and Smart I/O processor Out Of Band control.
- Historical - Log of all system events, Character mode screen image, and Serial Numbers.

Control of CDM functions is either intrinsic (i.e. CDM components act automatically to perform the function, such as when a fan fails, the remaining fan has its speed increased and the fan failure indicator is lit) or external (i.e. the CDM subsystem gets external input requesting the function). External functions can be initiated from either the system interface port by one of the P6 processors or through the RS232 serial CDM interface connected to the CDM External port.

The CDM subsystem remote access feature provides for remote management of the system when the Operating System is not available[2].

[2]See section on Remote Management.